14.–17. 09. 2009
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

WS-Policy

Ein Standard mit Schwächen

Bernhard Hollunder

Department of Computer Science
Furtwangen University of Applied Sciences

Agenda

- Web Services
- Quality of Service
- WS-Policy
  - Assertions and Operators
  - Runtime Behavior
- Applying WS-Policy
  - Increasing the expressivity of WSDL
  - Formal representation of Quality of Service (QoS) attributes
  - Service selection

Web Services – Basics

- Service interfaces are defined with WSDL
- Exchange of SOAP documents
- Platform independency
- Interoperability
- Extensibility towards Quality of Service

```
<s:Envelope ...>
  <s:Header>
    ...
  </s:Header>
  <s:Body>
    ...
  </s:Body>
</s:Envelope>
```

Separation of Concerns (I)

- Typically, a Web service implementation comprises
  - the core business functionality and
  - non-functional aspects such as security, fault tolerance, performance, scalability, …
- Functional and non-functional aspects should be separated:
  - increased reusability
  - reduced complexity of the implementation
  - the different aspects can evolve independently from each other
  - reduced maintenance efforts

Separation of Concerns (II)

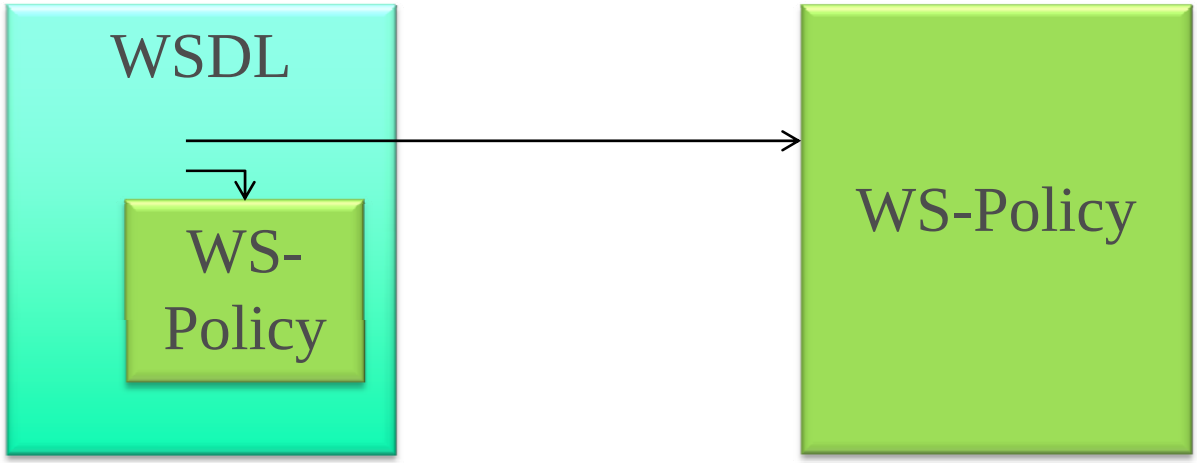
Typically in some programming language such as Java, C# ...

Ideally, specified in a declarative manner
→ Policy language

- The Web service's runtime environment is responsible for guaranteeing the specified behavior.

WS-Policy

- WS-Policy is a well-known framework for defining policies.
- W3C Recommendation: Web Services Policy 1.5 (Sep. 2007)
- Widely used and supported (Glassfish, Axis2, WCF, …)
- WS-Policy descriptions can be attached to WSDL.



WS-Policy: Assertions

- A policy is a collection of assertions.
- A single assertion may represent a capability, a requirement or a constraint.
- Every assertion has an XML representation. Examples:

```
<wsu:IncludeTimestamp/>
```

A timestamp has to be included in the message.

```
<sp:SignedElements>
  <sp:XPath>
    /S:Envelope/S:Body
  </sp:XPath>
</sp:SignedElements>
```

The body of the message must be signed.

```
<AccountingAssertion cost = "1"/>
```

The usage cost of the service is $1.

WS-Policy: Operators

- WS-Policy introduces operators to combine assertions.
- Example:

```
<wsp:Policy wsu:Id="SimplePolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:IncludeTimestamp/>
      <sp:EncryptSignature/>
    </wsp:All>
    <wsp:All>
      <sp:UsernameToken>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Alternative 1

Alternative 2

Runtime Behavior

- Suppose the server policy requires `IncludeTimestamp`.
- We can distinguish two cases:
  - Case 1: The client includes a timestamp. ✓
  - Case 2: The client does not include a timestamp. ✗

```
<s:Envelope ...>
  <s:Header> ...
    <o:Security> ...
      <u:Timestamp ...>
        <u:Created>2008-12-12T15:51:08.460Z</u:Created>
        <u:Expires>2008-12-12T15:56:08.460Z</u:Expires>
      </u:Timestamp>
    </o:Security>
  </s:Header>
  <s:Body>
    <sayHelloResponse ... >
      ...
    </sayHelloResponse>
  </s:Body>
</s:Envelope>
```

Case 1

WS-Policy: Summary

- WS-Policy defines a simple language:
  - Three operators: `All`, `ExactlyOne`, `Policy`
  - Normal form for policies
- Basic buildings blocks are assertions.
- Assertions have a
  - name (its QName)
  - zero or more nested child elements
  - zero or more assertion parameters
- The semantics of assertions are defined literally. Example:
  - “The SignedElements assertion is used to specify arbitrary elements in the message that require integrity protection.”

WS-Policy – Limitations

- Limited set of operators
  - No support for conditional assertions:

    „ The content of a Web service request must be digitally signed, if the request comes from a different security domain.”

- Standardized WS-Policy assertions are available only for few domains such as Message Security and Reliable Messaging.

- Policy intersection yields counterintuitive results:
  - When checking the compatibility of WS-Policy descriptions , the proposed algorithm of WS-Policy is not suitable in many situations.

Agenda

- Web Services
- Quality of Service
- WS-Policy
  - Assertions and Operators
  - Runtime Behavior
- Applying WS-Policy
  - Increasing the expressivity of WSDL
  - Formal representation of Quality of Service (QoS) attributes
  - Service selection

WS-Policy – OCL Case Study

- WSDL allows the definition of interfaces of Web services in an interchangeable, XML-based format.
- Among other information, WSDL introduces the name, the parameter types and faults of a Web service.
- This information is sufficient for a service consumer to construct a *syntactically* correct Web service request.
- However, WSDL does not allow the definition of arbitrary constraints imposed by the service implementation, e.g. range restrictions on parameters:
  - void f (int i) where (0 < i) AND (i < 200)

An Example: Weather Service

- There are providers for weather data (e.g., National Weather Service, NWS, http://www.nws.noaa.gov/).
- Typically, a weather service requires a longitute and latitude both of type xsd:decimal.
  - If called with 80 and -120, the caller gets following answer/exception: „Point is not on an NDFD grid“
  - If called with 37.8 and -122.4, the current weather in San Francisco is returned.

→The expressive power of WSDL can be increased with WS-Policy and other standard technologies.

Questions and Anwers

- How to formulate constraints in a standardized manner?

  → Object Constraint Language (OCL)

- How to associate WSDL entities with OCL constraints?

  → WS-PolicyAttachment and WS-Policy

- Syntactic representation of OCL expressions?

  → OCL-Assertions for WS-Policy

- How to adapt the runtime environment to check the constraints?

  →OCL-Handler for Web services infrastructures

Object Constraint Language (OCL)

- OCL is used in combination with the Unified Modeling Language (UML) to further constrain UML models in a clear and unambiguous manner.
- In OCL, “a constraint is a restriction on one or more values of (part of) an object-oriented model or system”.
- Textual representation of the constraint expressions.
- Important OCL constructs:
  - Preconditions and postconditions
  - Invariants
  - Collections

OCL Examples

- Precondition and postcondition

```
context Math::squareRoot(Real arg)
  pre:  arg >= 0
  post: result >= 0
```

- Invariant

```
context Customer
  inv: this.age >= 0 and
       this.address.zip.size() == 5
```

OCL-Assertions for WS-Policy

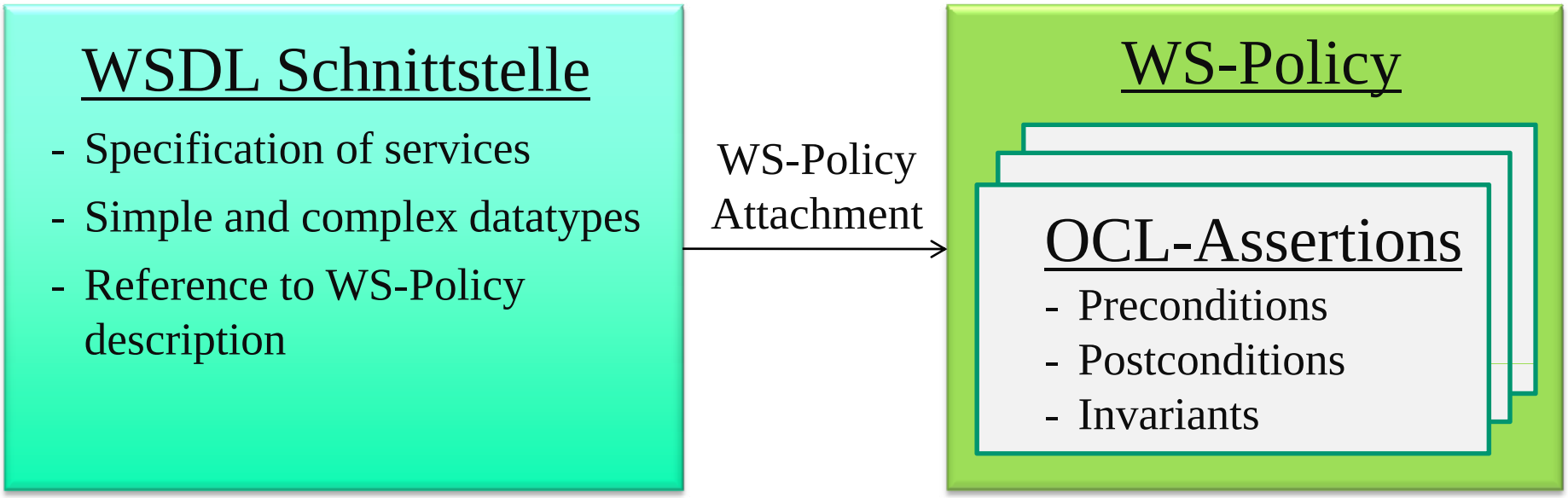
- OCL-OperationAssertion: specifies pre- and postconditions

```
<ocl:OperationAssertion Context = "NDFDgenByDay">
  <ocl:Precondition>
    longitude &gt; -130 AND longitude &lt; -59
  </ocl:Precondition>
  ...
  <ocl:Postcondition>
    result.size() &gt; 0
  </ocl:Postcondition>
</ocl:OperationAssertion>
```

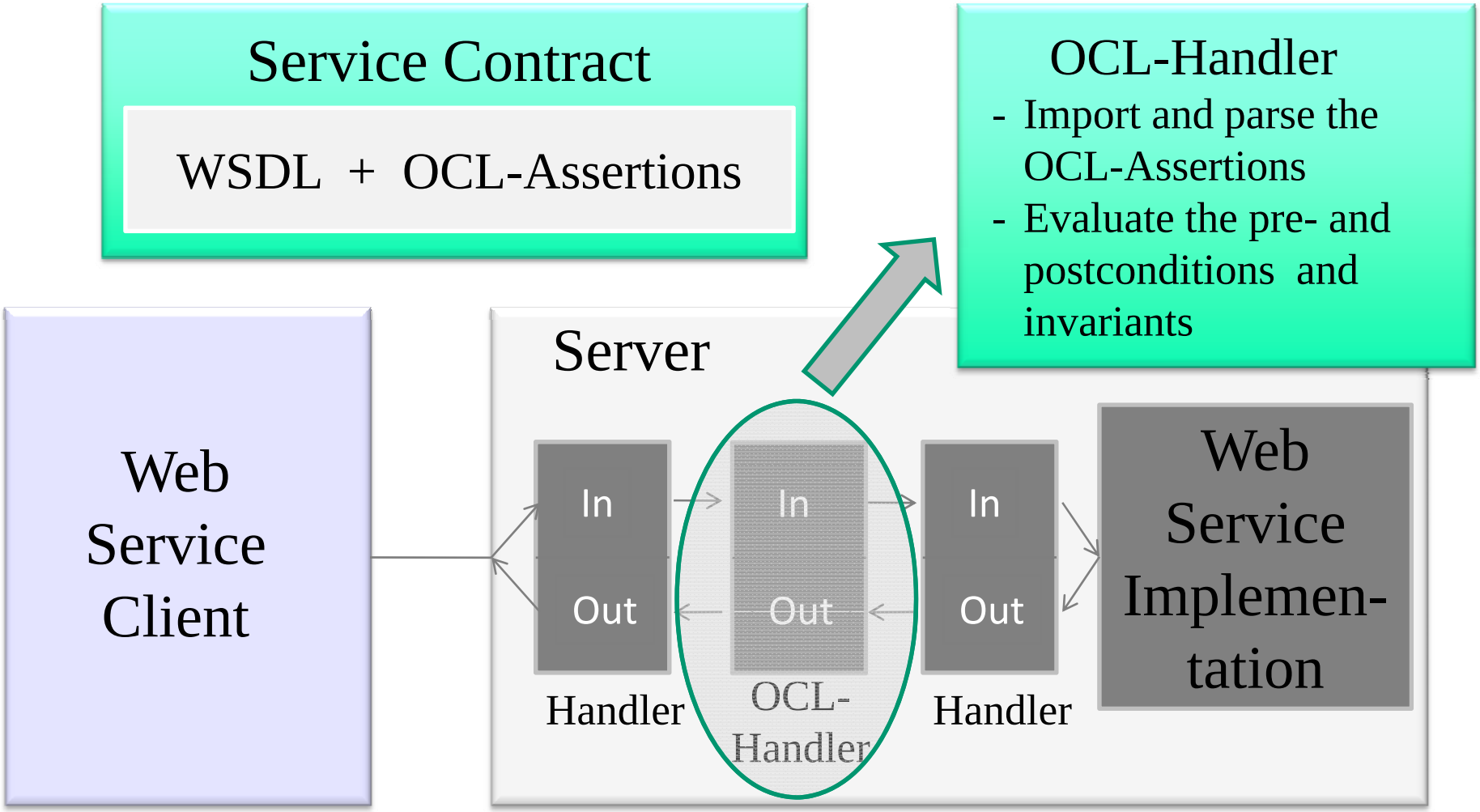
- OCL-DataAssertion: specifies invariants

```
<ocl:DataAssertion Context ="CustomerData">
  <ocl:Inv>
    this.age &gt;= 0
  </ocl:Inv>
</ocl:DataAssertion>
```

WSDL and OCL


WSDL Schnittstelle
- Specification of services
- Simple and complex datatypes
- Reference to WS-Policy description
WS-Policy Attachment
WS-Policy
OCL-Assertions
- Preconditions
- Postconditions
- Invariants

Runtime Environment



Agenda

- Web Services
- Quality of Service
- WS-Policy
  - Assertions and Operators
  - Runtime Behavior
- Applying WS-Policy
  - Increasing the expressivity of WSDL
  - Formal representation of Quality of Service (QoS) attributes
  - Service selection

Custom Assertions

- WS-Policy does not come with specific assertions.
- WS-* specifications (e.g. WS-SecurityPolicy) define domain-specific assertions such as `IncludeTimeStamp`.
- In many situations, it would be helpful to have a further repertoire of assertions covering specific themes such as:
  - Constraints on Web service parameters (→ OCL-Assertions)
  - A specified (average) response time will be guaranteed.
  - The usage of the service does not cost more than $1.
  - The usage costs of a Web service decreases by 10%, if the service is invoked between 10pm and 5am.

Custom Assertions: Formalization

1. Define the XML representation, e.g.

```
<PerformanceAssertion average_runtime = "1000"/>
```

```
<AccountingAssertion cost = "1"/>
```

```
<If type="TimeCondition" from="10pm" to="5am">
  <wsp:All>
    <DiscountAssertion discount="10%"/>
  </wsp:All>
</If>
```

2. Implement Java/C#/… classes that realize these assertions.
3. Extend the runtime environment such that the semantics of the assertions is respected.

Different Types of Assertions

- There are two types of custom assertions:
- Type 1:
  - A Type 1 assertion does not require that the Web service client includes additional information into the SOAP request.

  → Runtime environment must be extended only on *server* side.

- Type 2:
  - A Type 2 assertion can only be properly evaluated on server side, if the client includes further data into the SOAP header.

  → Runtime environment must be extended on *client* and *server* side.

Type 1 Assertion

- Example: OCL-Assertions
- Actions on client side:
  - The *standard Web service runtime includes* the parameter values of the service request into the body of the SOAP message.
  - There is no need to pass additional data to the server.
- Actions on server side:
  - The values of the parameters are extracted from the body of the SOAP message and passed to the OCL-Checker.
    - Constraint violation → rejection of the request
    - otherwise → delegation to the service implementation.

Type 2 Assertion

- Example: `<AccountingAssertion cost = "1"/>`
- Situation
  - Meaning of the assertion: The usage cost of the service is $1.
  - Suppose the client is willing to pay at most $0.5.

  → Contradictory requirement!
- Questions
  - Which part of the system is responsible for checking the condition?
  - Two answers:
    - Usage of a client policy.
    - Transmission of additional data to the server.

Type 2 Assertion: Client Policy

- Both server and client have their own WS-Policy descriptions.

Client Policy

```
<wsp:Policy wsu:Id="SimplePolicy">
 <wsp:All>
  <wsp:All>
   AccountingAssertion cost = "0.5"/>
  </wsp:All>
 </wsp:All>
</wsp:Policy>
```

Server Policy

```
<wsp:Policy wsu:Id="SimplePolicy">
 <wsp:All>
  <wsp:All>
   AccountingAssertion cost = "1"/>
  </wsp:All>
 </wsp:All>
</wsp:Policy>
```

- WS-Policy introduces policy intersection to check the compatibility of two policies.
- In this example: Client and server policies are not compatible.
- Requirement: Client must apply policy intersection before service invocation.

Type 2 Assertion: Transmission of Assertion Parameters

- Strategy
  - The client runtime environment puts assertion parameters into the header of the SOAP message.
  - These are retrieved and checked by the server runtime.

```
<wsp:Policy wsu:Id="ClientPolicy">
 <wsp:All>
  <wsp:All>
   AccountingAssertion cost = "0.5"/>
  </wsp:All>
 </wsp:All>
</wsp:Policy>
```

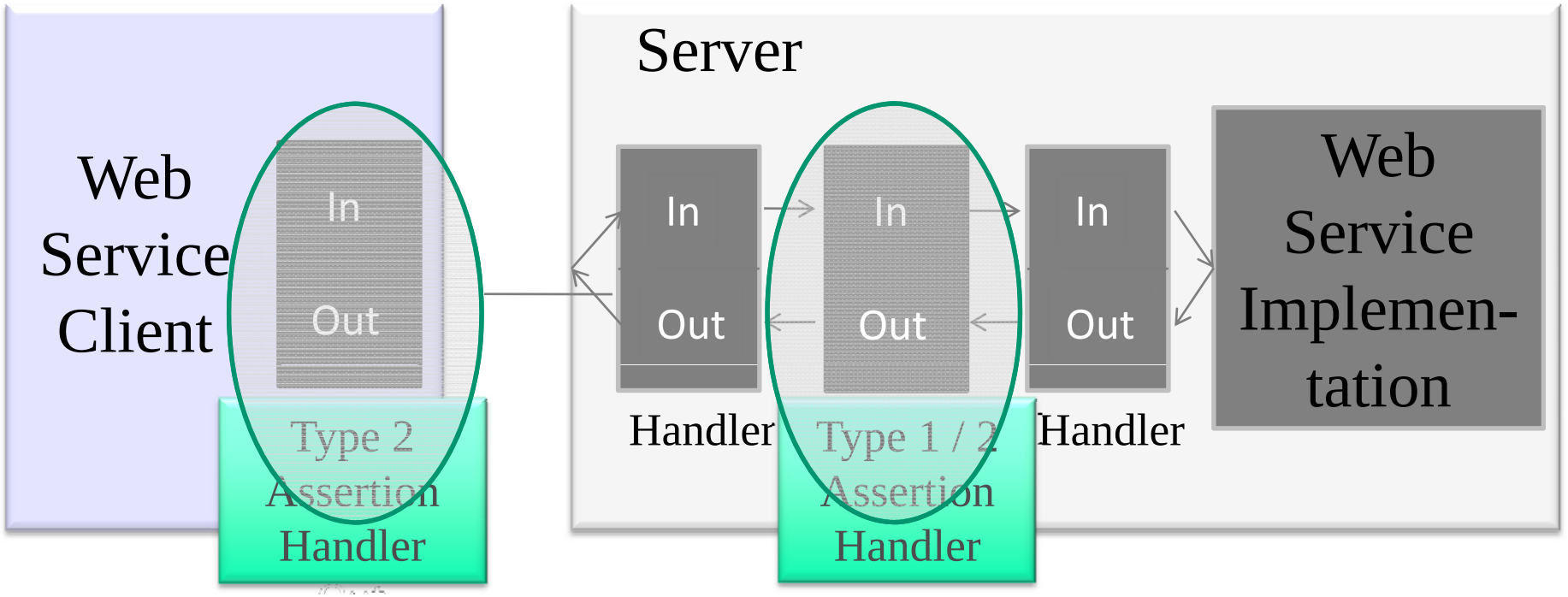
```
<wsp:Policy wsu:Id="ServerPolicy">
 <wsp:All>
  <wsp:All>
   AccountingAssertion cost = "1"/>
  </wsp:All>
 </wsp:All>
</wsp:Policy>
```

SOAP Request created by client runtime

```
<s:Envelope ...>
 <s:Header> ...
  <AccountingAssertion cost="0.5"/>
 </s:Header>
<s:Body> ... </s:Body>
</s:Envelope>
```

Contradiction detected by server runtime

Implementation Issues

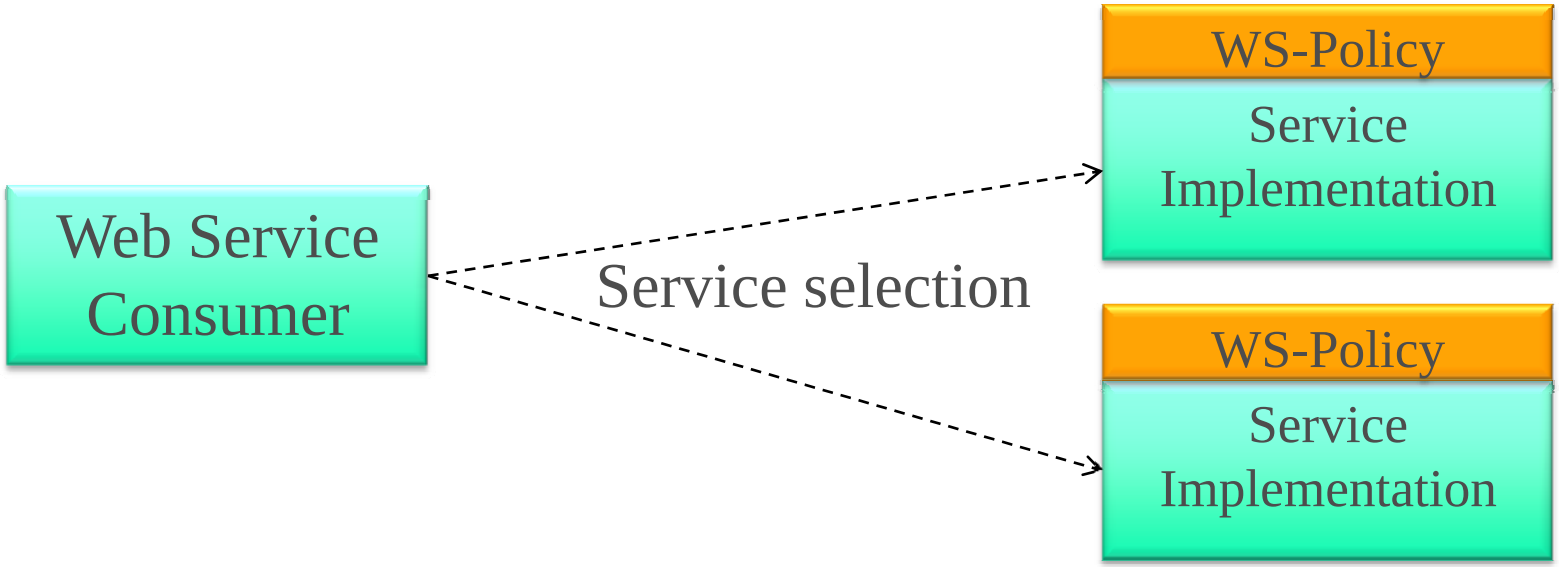


Agenda

- Web Services
- Quality of Service
- WS-Policy
  - Assertions and Operators
  - Runtime Behavior
- Applying WS-Policy
  - Increasing the expressivity of WSDL
  - Formal representation of Quality of Service (QoS) attributes
  - Service selection

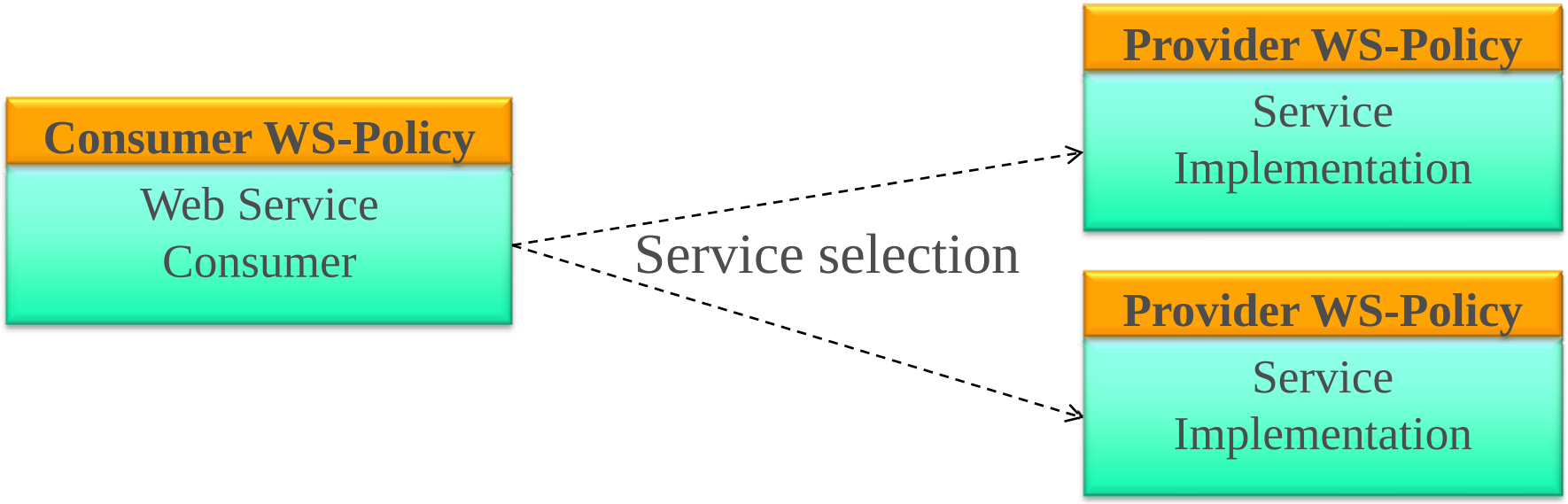
WS-Policy and Service Selection

- A WS-Policy description
  - expresses non-functional requirements in a formal manner
  - helps the Web service consumer to find an appropriate service.
- Problem: How to automate service selection?



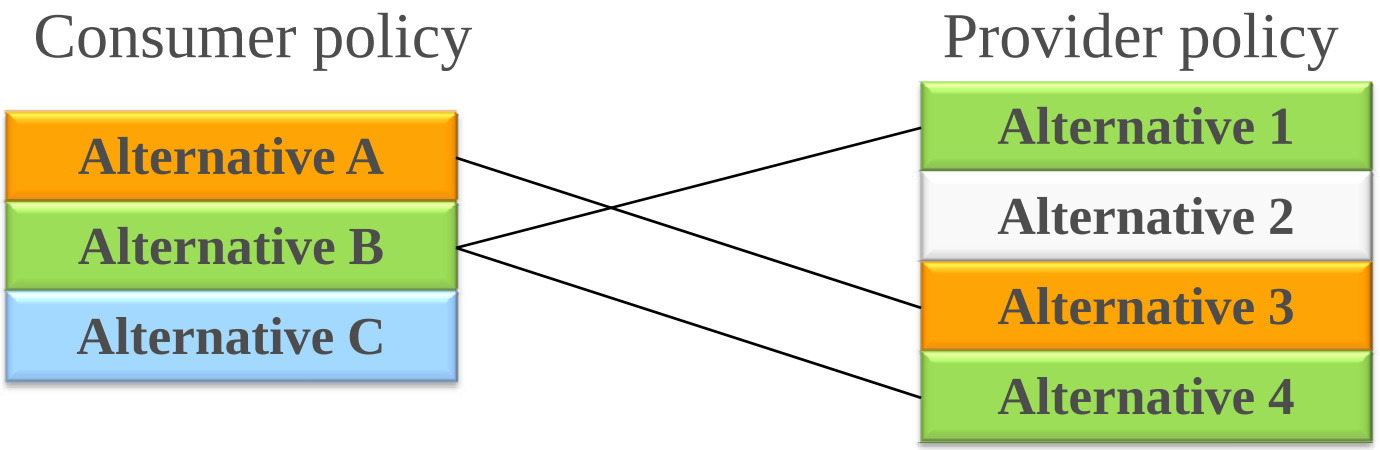
Consumer vs. Provider Policy

- Solution: Both parties have their own WS-Policy descriptions.
  - **Provider policy**:
    Conveys a set of conditions coupled with the usage of the service.
  - **Consumer policy**:
    Specifies the conditions under which he is willing to use the service



Policy Intersection

- WS-Policy defines policy intersection as a “first approximation” for determining the compatibility of policies.



A connection indicates compatible alternatives.

Policy Intersection: Examples

P1
```
<sp:SignedElements>
  <sp:XPath>/S:Envelope/S:Body</sp:XPath>
</sp:SignedElements>
```

P2
```
<sp:SignedElements/>
```

Compatible

P3
```
<wsam:Adressing>
  <wsp:Policy/>
</wsam:Adressing>
```

P4
```
<wsam:Adressing/>
```

Incompatible

P5
```
<AccountingAssertion cost = "1"/>
```

P6
```
<AccountingAssertion cost = "0.5"/>
```

Compatible

→ Policy intersection may produce counterintuitive results.

Policy Intersection: Problems

- Policy intersection
  - is a purely syntactic approach
  - does not incorporate the semantics of assertions
  - ignores parameters of assertions
  - synonymous assertions are considered incompatible.
- Currently, there is no uniform approach that combines both the domain-independent intersection and domain-specific processing.

→ Policy Entailment: new approach for checking the compatibility of WS-Policy descriptions

Policy Entailment: The Idea (I)

- Example:
  - Assertion A1: `<AccountingAssertion cost = "1"/>`
  - Assertion A2: `<AccountingAssertion cost = "0.5”/>`
  - A1 entails A2 because: if someone is willing to pay $1 for the usage of a particular service, then he is also willing to pay $0.5 (but not vice versa).

Policy Entailment: The Idea (II)

- Define a binary relation between the assertions, which reflects the intented semantics of the assertions, e.g.

  Assertion a: `<IncludeTimestamp/>`

  Assertion b: `<AccountingAssertion cost = "1"/>`

  Assertion c: `<AccountingAssertion cost = "0.5"/>`

  a < a, b < b, c < c, b < c

- Definition of Policy Entailment and Policy Differencing.
- Formal and algorithmic foundations are described in
  - Bernhard Hollunder: Domain-Specific Processing of Policies or: WS-Policy Intersection Revisited; International Conference on Web Services (ICWS 2009).

Summary

- WS-Policy is well-known and widely used standard.
- Framework for defining policies
  - Small set of operators
  - Extensibility through custom assertions
  - Integration with other technologies such as WSDL through WS-PolicyAttachment
  - Small set of standardized assertions
  - Counterintuitive results of policy intersection
- Multipurpose framework
  - Increasing the expressivity of WSDL
  - Formal representation of Quality of Service attributes through custom assertions

14.–17. 09. 2009
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Vielen Dank!

Bernhard Hollunder

Department of Computer Science, Furtwangen University of Applied Sciences